Sustainable Computing: Informatics and Systems 45 (2025) 101077

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Hybrid Markov chain-based dynamic scheduling to improve load balancing performance in fog-cloud environment

Navid Khaledian [a], Shiva Razzaghzadeh [b,*], Zeynab Haghbayan [b], Marcus Völp [a]

[a] *Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Esch-sur-Alzette, Luxembourg*
[b] *Department of Computer Engineering, Ardabil Branch, Islamic Azad University, Ardabil, Iran*

ARTICLE INFO

Keywords:
Task scheduling
Load balancing
Arithmetic Optimization Algorithm
Markov chain
Fog computing

ABSTRACT

Fog computing is a distributed computing paradigm that has become essential for driving Internet of Things (IoT) applications due to its ability to meet the low latency requirements of increasing IoT applications. However, fog servers can become overburdened as many IoT applications need to run on these resources, potentially leading to decreased responsiveness. Additionally, the need to handle real-world challenges such as load instability, makespan, and underutilization of virtual machine (VM) devices has driven an exponential increase in demand for effective task scheduling in IoT-based fog and cloud computing environments. Therefore, scheduling IoT applications in heterogeneous fog computing systems effectively and flexibly is crucial. The limited processing resources of fog servers make the application of ideal but computationally costly procedures more challenging. To address these difficulties, we propose using an Arithmetic Optimization Algorithm (AOA) for task scheduling and a Markov chain to forecast the load of VMs as fog and cloud layer resources. This approach aims to establish an environmentally load-balanced framework that reduces energy usage and delay. The simulation results indicate that the proposed method can improve the average makespan, delay, and Performance Improvement Rate (PIR) by 8.29 %, 11.72 %, and 4.66 %, respectively, compared to the crow, firefly, and grey wolf algorithms (GWA).

1. Introduction

The IoT sector has grown significantly in recent years, enabling the digitization of the physical world and connecting people and devices [1]. This rapid expansion has increased the demand for stability and low latency in IoT applications. Cloud computing, a critical component of IoT, provides the storage and processing power necessary to handle the vast amounts of data generated by IoT devices. However, the physical distance between cloud servers and IoT devices often results in high latency, making it unsuitable for real-time IoT applications [2]. Fog computing has emerged as a complementary paradigm to cloud computing to address these latency issues. Fog computing allocates processing and storage resources closer to IoT devices, significantly reducing communication time between IoT nodes and servers. However, the exponential increase in servers and IoT applications in fog computing environments introduces new challenges, such as the need for efficient execution and optimal load balancing [3]. Effective load balancing ensures that no single server is overwhelmed and that tasks are evenly distributed across available resources. This enhances server reliability and response times and improves overall system performance. Optimizing load balancing while minimizing response time is a complex but crucial task in fog computing environments. The fog-cloud architecture necessitates efficient task scheduling, as the level of service required by applications depends on their execution location within the cloud or fog. Fog infrastructure typically has lower processing power than cloud infrastructure, complicating resource management in the fog-cloud context. Efficient resource management and job scheduling policies can improve resource efficiency and reduce service times, creating a dependable communication environment [1].

Due to the increasing number of IoT tasks and the limited resources in fog computing, there often needs to be more load distribution. Managing computing, processing, and load distribution in a cloud environment is complex. Optimal task scheduling and efficient resource management are essential to distribute the load effectively. Previous research has used mathematical methods, machine learning, and meta-heuristic algorithms to reduce delays and predict traffic in fog nodes [4,

* Corresponding author.
E-mail addresses: navid.khaledian@uni.lu (N. Khaledian), Shiva.razzaghzadeh@srbiau.ac.ir (S. Razzaghzadeh), haghbayan.iauard@gmail.com (Z. Haghbayan), Marcus.voelp@uni.lu (M. Völp).

https://doi.org/10.1016/j.suscom.2024.101077
Received 7 August 2024; Received in revised form 16 December 2024; Accepted 18 December 2024
Available online 21 December 2024

5]. Directed Acyclic Graphs (DAGs) are frequently used to model IoT applications, with nodes representing tasks and edges representing data transfer dependencies. Task dependencies complicate scheduling, making heuristics and rule-based approaches insufficient for IoT application scheduling in fog computing environments [6,7]. In fog computing environments, computational resources are often limited, necessitating usinuselgorithms with lower computational complexity. The Arithmetic Optimization Algorithm (AOA) is particularly well-suited for this task scheduling problem due to its reliance on simple mathematical operations—addition, subtraction, multiplication, and division. This simplicity is crucial in scenarios where rapid processing and task scheduling are essential. Moreover, many traditional algorithms, such as genetic and simulated annealing, can become trapped in local optima while searching for solutions. AOA mitigates this issue by employing global search strategies, which enhance its ability to identify more optimal solutions for task scheduling. Another significant advantage of AOA is its compatibility with other optimization techniques, such as Markov chains. This synergy allows for the dynamic prediction of node loads and resource availability, enabling real-time load management and preventing overloads that could lead to performance degradation.

Makespan refers to the total time taken to complete a set of tasks, and a longer makespan can lead to delays in processing data, which is detrimental for real-time applications. Delays in the time taken for a task to be processed after submission can severely impact user experience, especially in critical applications like innovative traffic management. Additionally, PIR indicates the performance improvement achieved by a scheduling strategy; a low PIR due to overburdened servers signifies inefficiencies in task scheduling and resource allocation. Collectively, increased makespan and delay, along with a lower PIR, diminish the effectiveness and reliability of IoT applications, highlighting the need for effective task scheduling and resource management in fog computing environments.

New and highly effective optimization methods can be used to solve problems in graph theory, networks, path optimization, system design, and other areas of engineering and various sciences. Based on this, our proposed method also utilizes AOA for task scheduling. This paper addresses the challenges in fog computing, which is essential for IoT applications due to its ability to reduce latency by processing data closer to devices. As IoT applications grow, fog servers face resource strain, leading to reduced responsiveness. The need to manage issues such as load instability, makespan, and underutilization of VMs has driven the demand for more efficient task scheduling in fog and cloud environments. The significance of this paper lies in developing cost-effective and efficient scheduling solutions that balance the computational load, reduce delays, and lower energy consumption in fog systems with limited resources. This paper proposes using the AOA for task scheduling and Markov chains to forecast VM load. The goal is to create an optimized and balanced framework that improves system efficiency by reducing energy usage, delays, and makespan, providing a practical solution for managing the growing IoT loads in fog and cloud environments. In conclusion, this paper offers an innovative approach to optimizing IoT application management and scheduling in fog computing, enhancing performance under resource constraints.

The paper is divided into the following sections: In the second section, we provide an overview of the research on task scheduling and load balancing. Our suggested approach is presented in the third part. The fourth section describes the simulation and the outcome of the proposed strategy. In section five, we finally offer the article's conclusion.

2. Related work

Hudson et al. [8] proposed a method that addresses the challenge of AI-based service placement and request scheduling in edge computing to maximize Quality of Service (QoS). The method formulates the problem as an Integer Linear Program (ILP), demonstrating that placement and scheduling are NP-hard. Additionally, the proposed FLIES algorithm for predicting future requests and a collaborative placement algorithm (CGP) outperform existing approaches in maximizing QoS. Another method has proposed resource allocation with efficient task scheduling based on a hierarchical auto-associative polynomial convolutional neural network [9]. This method reduces makespan and increases throughput using a short-term scheduler based on the Starling Murmuration Optimizer (SMO). Efficient resource allocation in this method is based on design constraints such as bandwidth, resource load, and more. The authors in [10] address the problem of load balancing for virtual machines in cloud computing. This method fairly distributes tasks across virtual machines, considering the overall performance of the entire cloud. To solve this problem, the proposed method transforms it into an assignment problem and solves it using the Hungarian method. A Markov Decision Process (MDP) strategy was used by Liu et al. [11] to reduce average task execution delay in edge computing settings. They suggested a productive one-dimensional search method to identify the best work scheduling strategy. Nevertheless, expanding this work to address intricate weighted cost optimization problems in heterogeneous fog computing systems is challenging, and it cannot adjust to changes in the computing environment. To minimize the overall computation time and system cost of job scheduling in heterogeneous fog cloud computing systems, Ali et al. [12] suggested a Non-dominated Sorting Genetic Algorithm) NSGA II (-based method.Through their work, the task scheduling problem is formulated as an optimization problem to dynamically allocate suitable resources for predefined tasks. The authors introduced a novel HDSOSGOA technique based on a hybrid chaotic algorithm for work scheduling in fog computing settings. Grasshopper Optimization Algorithm (GOA) and Symbiotic Organisms Search (SOS) algorithms are used by HDSOSGOA, which selects amongst them depending on the probability calculated by the learning automata. The article aimed to reduce scientific workflows' makespan, cost, and energy consumption on cloud servers and fog nodes. The dynamic voltage and frequency scaling (DVFS) method is also used in the article to reduce energy usage. The best DVFS-level VMs are chosen using the HDSOS-GOA algorithm, which also assigns the jobs to the most suitable VMs. This article aims to reduce workflow scheduling time and energy consumption. The study also uses fog nodes close to end users to reduce latency and bandwidth usage [13]. Singh et al.'s paper [14] offered a hybrid GA-modified Particle Swarm Optimization (PSO) approach for resource assignment. In cloud-fog computing contexts, the strategy tries to minimize the makespan, cost, and energy consumption of tasks that depend on one another on different resources. To avoid job execution delays, the approach uses slow virtual machines. To do this, the method employs the same weights for the makespan, energy consumption, and execution cost in the fitness function. The study demonstrates that the suggested method outperforms existing algorithms regarding the makespan, cost, and energy consumption of executing scientific workflows using cloud and fog resources. To improve performance and cost-effectiveness, a unique scheduling strategy for complex applications with many jobs within edge cloud systems was developed in a study by Khaledian et al. [15]. This method seeks to optimize workflow scheduling in fog-cloud situations. It is based on the krill herd algorithm, a meta-heuristic algorithm inspired by the collective behaviour of krill swarms. Furthermore, combining dynamic voltage and frequency scaling (DVFS) approaches further improves energy efficiency by modifying fog nodes' voltage and frequency levels.To manage the load and decrease task execution latency in a fog cloud computing environment, Shahidani et al. [16] devised a Q-learning-based approach. Nevertheless, this work does not consider the inter-task dependencies and the heterogeneity of cloud and fog computing environments.A multi-objective cost-aware scheduling algorithm was proposed in the literature to solve the execution of bag-of-tasks (BoT) applications running on fog computing environments [17]. The proposed algorithm utilizes a load balancing algorithm to offload tasks of overloaded fog nodes into under-loaded ones to reduce *makespan* in favour of users. However, other suggested operators enhance other problems' objectives. A QoS-aware scheduling algorithm deploys the components of IoT applications on available fog nodes to increase running reliability and reduce power consumption [18]. Authors in [19] propose a new load balancing model for cloud environments, including a PSO algorithm, a VM migration model using Double Deep Q Proximal, and a GAN-based feedback controller to enhance fault tolerance, reduce energy consumption, and decrease migration time. To this end, the authors presented a resource dissipation model to assign tasks on processing nodes that minimize resource wastage. The "Load Balancing Aware Task Selection and Migration" (LBATSM) approach in [20] explores a novel method to enhance load balancing and efficiency in fog computing systems. This method comprises two main components: task selection and task placement. In the task selection phase, tasks are chosen for migration based on criteria such as the current load of nodes and task priorities. In the task placement phase, a Modified Binary Particle Swarm Optimization (MBPSO) algorithm is employed, enhanced with an improved transfer function to maintain a balance between exploration and exploitation. In [21], the authors introduce a new algorithm called DALBFog (Deadline-Aware and Load-Balanced Task Scheduling) for task scheduling in fog computing environments for IoT applications. The primary objective of this algorithm is to enhance task scheduling by simultaneously considering deadlines and load balancing among fog nodes. This approach utilizes two sub-schedulers: one for the initial allocation of tasks and another for rescheduling tasks that fail to meet their deadlines. Results demonstrate that the DALBFog algorithm outperforms existing methods in improving load balancing and meeting deadlines. This algorithm optimally allocates tasks to fog nodes, thus enhancing overall system efficiency. The approach in [22] presents a novel hybrid method for load balancing and fault tolerance in fog computing. This method combines the Ant Colony Optimization (ACO) algorithm and the Multi-Objective Hawk Optimization (MHHO) algorithm. The MHHO-ACO approach is used to distribute the load among fog nodes and execute redundant instances of applications on different nodes to ensure fault tolerance. This algorithm improves execution time, cost, and energy consumption.[23] investigates a cloud load balancing framework in fog computing environments. It introduces and analyzes a new model called AELBA (Advanced Ensemble Load Balancing Approach), which ensures resource utilization and improves the performance of fog computing systems by leveraging the collective intelligence of multiple load balancing and optimization algorithms. AELBA combines several load-balancing methods, such as dynamic threshold-based load balancing, round-robin scheduling (RR), and intelligent workload prediction. This approach enhances response time and reduces latency. Ghorbannia et al. proposed scheduling methods for improving energy, cost, throughput, and latency in cloud environments. Their research concentrated on workflows involving several tasks with priority in execution. Each task in the workflow was mapped to a virtual machine, considering optimization problems such as cost and energy. Their heuristic methods outperformed their comparing algorithms [24,25]. Deadline and Budget constrained Archimedes Optimization Algorithm (AOA) is designed for task scheduling in cloud computing, optimizing makespan and cost while adhering to deadline and budget constraints [26].

While existing algorithms offer appropriate techniques for scheduling tasks and allocating resources, most need to consider these two issues simultaneously. In addition to addressing these issues, our suggested approach presents a load-balancing architecture. It analyzes and quantifies the VM load using the Markov chain and then uses the AOA method to schedule tasks. This method lowers makespan, delay time, and energy usage.

3. Proposed method

By connecting devices and apps, the IoT turns massive amounts of different data into information that can be used. IoT devices frequently rely on cloud computing, which has excellent processing capacity but inherent delay due to computational and energy constraints. Fog computing lessens this, which moves processing closer to IoT devices at the network edge to reduce latency and boost productivity. It dramatically reduces the time data must travel between IoT nodes and servers for processing by using tiny, low-capacity computing devices before transferring the data to the cloud. The approach for task scheduling and resource management in fog and cloud layers presented in this chapter uses optimization techniques to reduce scheduling costs, time, and Markov chains to assess node loads.

3.1. Three-layer structure

This study proposes a dynamic resource allocation and task scheduling strategy to facilitate dynamic load balancing across all edge and cloud computing nodes. Fog computing moves processing capabilities closer to IoT devices, enhancing service delivery for IoT applications. The proposed method provides a practical scheduling and resource management approach in the fog and cloud layers. Understanding IoT architecture is crucial for resource management and task scheduling. IoT consists of three layers in its simplest form: Sensor Layer: Generates tasks from users or smart devices, initially entering the fog layer for scheduling and resource allocation. Fog Layer: Receives requests from the sensor layer and requires mechanisms for utilizing fog resources for processing and scheduling tasks. Cloud Layer: Handles incoming requests from the fog layer; for real-time tasks, both fog and cloud resources are utilized [27]. We have considered a resource management and task scheduling system in the fog layer. This system operates as follows: A set of tasks is generated from the sensor layer and sent to the fog layer for scheduling. Fog resources sometimes need help to handle a large volume of user requests, causing load imbalance. Therefore, load balancing among all resources in the fog layer is necessary. The tasks sent to the fog layer are handed over to the load balancer and scheduler. The load balancer receives the tasks and determines their real-time or non-real-time nature based on their deadlines. Real-time tasks are processed in the fog layer, while non-real-time tasks utilize resources from the fog and cloud layers. Two states (busy and idle) are considered for each VM to improve scheduling and achieve the predicted goals when calculating the load.

Additionally, a Markov chain is used to predict the status of a VM (in terms of high or low load) or related resources. If a VM has a high load, tasks are removed and executed on a VM with a lower load. The load balancer sets an objective function for task scheduling and completes tasks on available resources to optimize this objective function. In the proposed method, the AOA is used to reduce the execution time of tasks. Fig. 1 illustrates an example of the three-layer architecture. Table 2 shows the notations used in this paper. Attempts have been made to use standard notations.

3.2. Resource allocation with Markov process

A Markov model is a set of probabilistic functions of states. In this context, the resulting model is a stochastic model with an underlying random process that is hidden and can only be observed through a set of random processes generating the sequence of observations. In a Markov model, the following parameters can be established: Number of Possible States: The number of states plays a crucial role in the model's success, with each state corresponding to a specific event. Various methods exist for connecting states; in the most general form, all states are interconnected and accessible. Number of Observations in Each State: The number of discrete observations equals the number of outputs the modelled system will have. If the discrete observations have an unlimited value, Eq. (1) describes it.

$$A = \{a_{ij}\}\ ,\ a_{ij} = \mathrm{p}\{q_{t+1} = \mathrm{j} | q_t = \mathrm{i}\}, \quad 1 \le \mathrm{i}, \mathrm{j} \le N; \tag{1}$$

Where q_t presents the system's state at time t, indicating the current

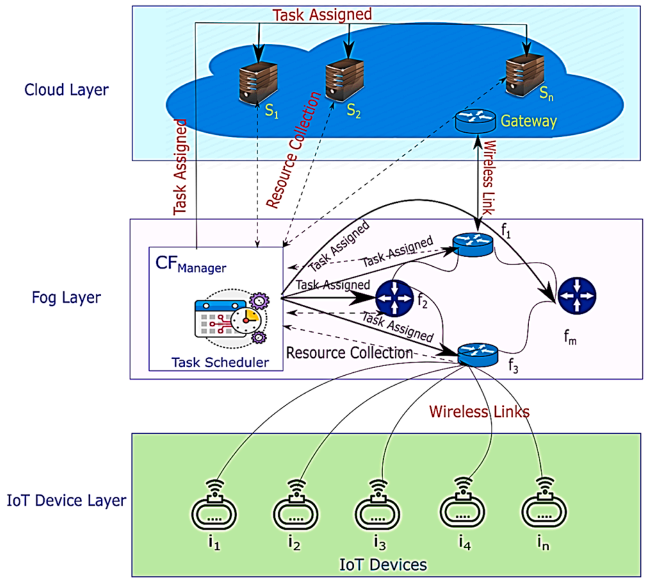

Fig. 1. three-layer architecture [28].

state of the system. N denotes the total number of nodes or states in the system. t refers to a specific time when the system is in a particular state. a_{ij} is the probability of transitioning from node i to node j, representing the likelihood of the system moving from state i to state j at a given time. Finally, i and j represent the indices of the different states of the system, each referring to a specific state within the model. These constraints are included in Eq. (2).

$$a_{ij} \geq 0, \quad 1 \leq i,j \leq N, \sum_{j=1}^{N} a_{ij} = 1, \quad 1 \leq \mathrm{i} \leq \mathrm{N} \tag{2}$$

For the states of an ergodic model, $a_{ij} > 0$ for all i *and* j. In the case where there is no connection between states $a_{ij} = 0$.Generally, there are two methods for load balancing: one is through data migration across different links, and the other is using evaluation algorithms to manage the load of VM systems. It is assumed that the system's current state depends on previous states. Based on the request load and threshold value, the local computational method decides whether to send a request to the central server or neighbouring network servers for processing. This method ensures that servers operate within a reasonable load range. Assume that $ss = \{1, \quad 2, \ldots, \mathrm{N}\}$ specifies the set of all virtual machines and three threshold levels of high, standard, and low load are denoted as MAX(i), NORMAL(i)and MIN(i), respectively. Two characteristics mainly represent the utilization criteria of individual services for virtual machines. E(t) denotes execution time, W(t) denotes job waiting time, and the weight parameterλ is formulated according to Eq. (3). The input is obtained from the dynamic hierarchical weighting design method, and the load characteristic of task i at time t is L(t):

$$L_i(t) = \quad \lambda_1 E_i(\mathrm{t}) + \quad \lambda_2 W_i(\mathrm{t}) \tag{3}$$

Where $\Sigma\lambda i = 1$. Additionally, the overall load status of the VM at time t can be specified by the parameter LOAD(i), as shown in Eq. (4).

$$\mathrm{LOAD(i)} = \sum_{i=1}^{n} L_i(t) \tag{4}$$

The centralized scheduling policy configures a module to collect the load status of each node, adjusting the allocation of dedicated service resources of the nodes according to the hierarchical task requirements within the cloud framework (Fig. 2).

This scheduling program unifies resources for multi-task combinations of task sets and does not accept new tasks for processing when

Table 1
Comparison of Existing Approaches.

Ref	Main idea	Advantages	Disadvantages
[8]	optimize AI service placement and request scheduling in edge computing	Maximizes QoS, Collaborative Placement (CGP), Scalable and Efficient.	Complexity, FLIES depends on decentralized data
[9]	improve resource efficiency and reduce response time in cloud systems with Convolutional Neural Network	increases resource utilization, reduces energy consumption, improves response time	Not justifying the computational cost, lack of comprehensive evaluation and security considerations
[10]	transform the cloud computing load balancing problem into an allocation problem and solve it using the Hungarian method	Improvement in makespan and throughput, decrease in VM utilization, Deviation	the time complexity of this algorithm could become problematic, and Limited Evaluation Metrics (Makespan and throughput)
[11]	proposes an effective solution for optimizing task scheduling in Mobile-Edge Computing systems	Reducing overall system latency, Optimizing the use of resources, Reducing energy costs	Complex implementation, lack of attention to interactions between tasks, lack of practical assessment
[12]	Designing an automated task scheduling model for Fog-Cloud systems using the NSGA-II.	Multi-objective optimization reduces processing delays and increases the productivity of resources, flexibility and scalability.	High computational complexity, need for fine-tuning, lack of attention to security and privacy, and limitations in practical evaluation.
[13]	Designing a scheduling algorithm for workflows in Fog Computing systems Using a chaotic hybrid algorithm (Hybrid Chaotic Algorithm)	Optimizing energy consumption, efficient resource management, reducing cloud and edge systems load, and reducing costs.	High computational complexity, Need to fine-tune the parameters, Limitation in scalability, lack of attention to security and privacy issues.
[14]	Combining PSO and GA algorithms for optimal task scheduling in cloud-fog environments	Multi-objective optimization, reducing delay time, Reducing costs, and optimizing energy consumption.	High computational complexity, Need to fine-tune the parameters, time-consuming for large scales, lack of attention to security and privacy issues.
[15]	Designing an improved method for workflow scheduling in fog-cloud environments using the Krill Herd algorithm.	Improve energy consumption, fault tolerance, multi-objective optimization, system efficiency, scalability and flexibility.	High computational complexity, Need to fine-tune the parameters, Constraints in the real world, and Scalability problems at large scales.
[16]	Using a reinforcement learning algorithm, a multi-objective method to optimize task scheduling and load distribution in edge-fog-cloud architecture was presented.	Reduce delay time, multi-objective, reduce delay time Optimize energy consumption, and load balancing.	High computational complexity needs fine-tuning, scaling, and dependence on input data.
[17]	Modeling the scheduling of tasks optimally in the fog environment to reduce costs and improve performance	Multi-objective Optimization, Cost-aware model, Improvements in scalability and efficiency, Flexibility in resource allocation	Complexity of algorithms and implementation, Limitation in scalability of algorithms, Dependence on accurate input data, Failure to address security and privacy complexities
[18]	Multi-objective and QoS-aware optimization for the deployment of IoT	Improving QoS in distributed environments, High	High complexity and computational cost, dependence on accurate

(continued on next page)

Table 1 (*continued*)

Ref	Main idea	Advantages	Disadvantages
	applications in "fog-cloud infrastructures	scalability, Reducing energy consumption	data, lack of attention to security and privacy
[19]	Load balancing model for cloud computing using PSO, Double Deep Q Proximal for VM migration.	Optimized Load Balancing, Efficient VM Migration, Energy Efficiency	High Complexity, Initial Cost
[20]	Load balancing, selecting and shifting tasks in the fog computing	Increase system efficiency, Reduce problems caused by overloading, Reduce processing delays, and ability to run in real-time	High computational complexity, need for accurate data, Limited flexibility against rapid load changes, Scalability issues
[21]	Designing a time-sensitive scheduling algorithm (Deadline) and load balancing in Fog computing environments for IoT applications	Reduce latency, Increase efficiency, Optimal scheduling and aware of the time limit, load balancing on Fog nodes, scalability in changing environments	Limited flexibility against dynamic conditions, Reduced efficiency in critical or complex situations, Dependence on Fog infrastructure, High computational complexity
[22]	Use of advanced algorithms to divide the load and identify failures effectively	Improve performance with load balancing, increase fault tolerance, reduce latency, Optimize the use of resources, and improve system security and stability.	Dependency on specific Fog infrastructure, Scalability problems in large environments, lack of attention to security issues, Failure to consider energy issues
[23]	Providing an advanced hybrid approach for load balancing management in Fog computing using different algorithms simultaneously computing	Reduce latency, Use ensemble approach, Resource optimization, High scalability and flexibility, and Optimal load distribution.	High computational complexity, Scalability problems in large environments, Failure to consider energy consumption
[24]	Optimize the scheduling of cloud workflows, especially for resource-intensive IoT applications based on distance parameters, input clustering, and real-time execution times.	Optimizing energy consumption and cost, Improved processing power and latency, Reducing migrations of virtual machines	Implementation complexity, Lack of scalability, Failure to investigate environmental impacts
[25]	Using the migration of virtual machines and combined parameters for energy consumption and cost	Reduce energy consumption and cost, Improve efficiency, flexibility and usability, Resource optimization	Complexity of implementation and execution, Limitation in scalability, Needs fine-tuning
[26]	workflow scheduling in the cloud with Archimedes Optimization Algorithm	Simultaneous management of time and budget constraints, improving resource efficiency, reducing costs by complying with restrictions, Reducing latency	High computational complexity, Limitations in scalability in certain situations, requires high computing resources to run

virtual machines are overloaded. Regularly, there are multiple paths between nodes in the source network. If more connectable paths exist between nodes, it indicates a higher likelihood of choosing another route instead of the direct one, with a corresponding lower weight. The network topology is represented by an undirected graph $G = (V, E)$, where V represents the set of nodes in the network and E represents the set of links. Fig. 3 shows n paths from node i to j, and only P1 is a directly connected channel. The n-step transition probability expression under the Markov process $P^{(n)}$ calculates the fully connected adjacency matrix graph.

Table 2
Notations used in the formulae and method.

variables	Variables description
q_t	System status in time t
N	Total number of nodes
t	Time
a_{ij}	Probability of transition from node i to node j
E(t)	execution time
W(t)	waiting time
λ	weight parameter
$L_i(t)$	load characteristic of task i at time t
LOAD(i)	overall load status of the VM at time t
$P_{ij}^{(t)}$	conditional probability distribution of the state space from node i to node j
X_{n+1}	the state of the space node
x_n	state at time n
$P^{(n)}$	fully connected adjacency matrix graph
P_{ij}	Probability of accessing the link from node i to node j
T_{len}^{i}	duration of each task
VM_{Pow}^{j}	processing speed of tasks on a VM
N_{VM}	Number of virtual servers
NPH	Number of physical servers
ETC_{ij}	execution time of task i on virtual machine j
N_T	number of tasks entered into the fog layer
N_{VM}	number of virtual machines in the fog layer
D	delay
E	energy
α	random coefficient between zero and one
MIN	The minimum acceleration of the solution movements
MAX	The maximum acceleration of the solution movements
M_Iter	maximum number of iterations
MOP	mathematical optimization algorithm
LB_j	the lower bounds of the j -th dimension of a solution
UB_j	the upper bounds of the j -th dimension of a solution
μ	control parameter
ε	minimal constant value
$ExeTime(t_i)$	execution time of a task
PIR	percentage of performance improvement for each method
Y'	fitness obtained by the proposed method in scheduling

The one-step transfer matrix of the system state includes the one-step transition probability. The parameter $P_{ij}^{(t)}$ presents the conditional probability distribution of the state space, specifically the likelihood of transitioning from node i to node j at time t. The indices i and j refer to the node counters, representing specific states in the system. Xn + 1 denotes the state space node at time n + 1, while $\mathbf{x_n}$ represents the system's state at time n. t indicates a specific time point in the system's evolution. Lastly, $P^{(n)}$ is used to calculate the fully connected adjacency matrix graph, which defines the connections and transitions between nodes in the system. Therefore, the transition probability of reaching j from state i at time t is given by Eq. (5).

$$P_{ij}(t) = P\{X_{n+1} = j | x_n = i\} \tag{5}$$

Since link access is proportional to the sum of non-one-step transition probabilities of nodes i and j, it is rewritten as Eq. (6):

$$P_{ij} = \sum_{t=2}^{n} \left(P_{ij}^{(t)} + P_{ji}^{(t)}\right) \Big/ 2 \tag{6}$$

Therefore, when $P_{ij} \le MIN(d_i)$, it can be concluded that the VM is in a low-load state and can perform coordination tasks. When $P_{ij} \ge MAX(d_i)$, it can be judged that the VM is overloaded, and user response is delayed. So, necessary tasks can be directed to underloaded VMs until they converge to NORMAL(i). It is worth noting that d_i is the distance from the ith node. When the load is within the normal load range, virtual machines operate in a normal state, and the load in the cloud area can be sorted in ascending order as backup virtual machines to balance the load.

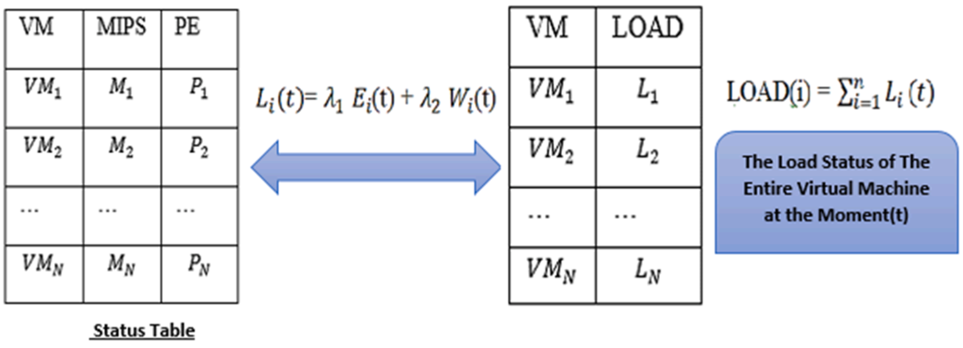

Fig. 2. Load status collector module of each node.

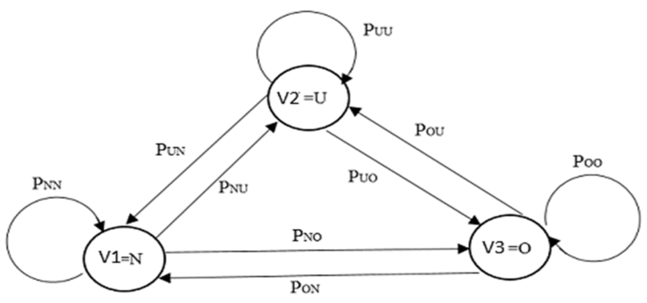

Fig. 3. Markov model in virtual machines.

3.3. Scheduling objective function in allocated resources in the fog layer

When tasks are scheduled only in the fog layer, it is assumed that there are Nph physical servers, and each physical service has N_{VM} virtual servers. The duration of each task can be represented by T^{i}_{len}. VM^{j}_{Pow} indicates the processing speed of tasks on a VM, with i as the task counter and j as the VM counter. The objective function for scheduling in the fog layer uses the proposed tasks in Eqs. (7), (8), and (9)[29].

$$ETC_{ij} = \frac{T^{i}_{\text{len}}}{VM^{j}_{\text{Pow}}}, i = 1, 2, \ldots, N_T, j = 1, 2, \ldots, N_{VM} \tag{7}$$

$$ETC = \begin{bmatrix} ETC_{11} & ETC_{12} & \ldots & ETC_{1N_{VM}} \\ ETC_{21} & ETC_{22} & \ldots & ETC_{2N_{VM}} \\ \ldots & \ldots & \ldots & \ldots \\ ETC_{N_T1} & ETC_{N_T2} & \ldots & ETC_{N_TN_{VM}} \end{bmatrix} \tag{8}$$

$$\text{Cost} = \min\left(\max_{j \in 1,2,\ldots,m} \left(\sum_{i=1}^{n} ETC_{ij} \right) \right) \tag{9}$$

ETC_{ij} is the execution time of task i on VM j (i is task counter and j is VM counter). N_T is the number of tasks entered into the fog layer, and N_{VM} is the number of VMs in the fog layer. The matrix ETC shows the execution time of tasks on the virtual machines in the fog layer. Eq. (9) is the objective function in the fog layer. The proposed method aims to schedule real-time tasks using a mathematical optimization algorithm to minimize the desired objective function.

3.4. Scheduling objective function in allocated resources in fog and cloud layers

If the task type is not real-time, task scheduling is executed in both the cloud and fog layers, and the cost function for task scheduling is presented in Eq. (10) [30].

$$F = \alpha \times D + (1 - \alpha) \times E \tag{10}$$

In this equation, D and E, respectively, represent the execution delay of the task in the fog and cloud layers and the amount of energy consumed during task scheduling. α is a random coefficient between zero and one [31].

3.5. Optimization of resource allocation for scheduling

The mathematical optimization algorithm is a meta-heuristic method for finding the optimal solution based on addition, subtraction, multiplication, and division operators. The mathematical optimization algorithm is used in this research for the following reasons: High accuracy, low complexity, exploration search, and exploitation search. The exploration and exploitation search of the mathematical optimization algorithm to find the optimal solution is shown in Fig. 4. The exploration search in this algorithm aims to reject local optimal solutions and find the global optima. In contrast, the exploitation search reduces the computational error of the global optimum. In this algorithm, mathematical operators are used for international or exploration search and local or exploitation search, as shown in Fig. 4: Addition and subtraction operators are used for the exploitation search. Multiplication and division operators are used for the exploration search.

In the mathematical calculation optimization algorithm, MOA and MOP parameters are used in order and according to Eqs. (11) and (12) in each iteration to update the solutions and determine the search strategy:

$$MOA(C\_Iter) = Min + C\_Iter \times \left(\frac{Max - Min}{M\_Iter}\right) \tag{11}$$

$$MOP(C\_Iter) = 1 - \frac{C\_Iter^{\frac{1}{\alpha}}}{M\_Iter^{\frac{1}{\alpha}}} \tag{12}$$

In these equations, Max and Min represent the maximum and minimum acceleration of solution movements and feature vectors to approach the optimal solution, mapping tasks to resources. The

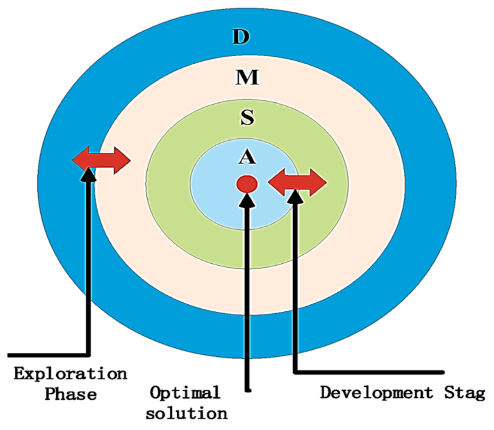

Fig. 4. Quadruple operators in mathematical calculation algorithm [23].

parameter *C_Iter* is the iteration counter of the mathematical optimization algorithm, and *M_Iter* is the maximum number of iterations considered for the scheduling and resource allocation algorithm. The value of the MOP coefficient in the mathematical optimization algorithm is determined in each iteration based on the α parameter. The α coefficient can be determined through trial and error at around 5.The MOA parameter is a metric for evaluating the algorithm's performance. Eq. (13) is used for exploration search, and in this case, a random number between 1 and 0 is generated. If this random number is less than 0.5 for each mapping of tasks to resources, the division operator is used to update the mapping of tasks to resources. Otherwise, the multiplication operator is used to update the mapping of tasks to resources:

$$x_{i,j}(C_{Iter}+1) = \begin{cases} best(x_j) \div (MOP+\varepsilon) \times ((UB_j - LB_j) \times \mu + LB_j) & rand > 0.5 \\ best(x_j) \times MOP \times ((UB_j - LB_j) \times \mu + LB_j) & rand \leq 0.5 \end{cases} \tag{13}$$

In this equation, $best(x_j)$ is the optimal solution or mapping of tasks to resources, and $UB_j$and $LB_j$are the upper and lower bounds of the j -th dimension of a solution, respectively. μ is a control parameter, and ε is a minimal constant value [32]. Rand () determines the type of search, local or global. If rand () is more significant than MOA, the search will be local or global. For updating the task mappings based on exploitation equations, Eq. (14) is used.

$$x_{i,j}(C_{Iter}+1) = \begin{cases} best(x_j) - MOP \times ((UB_j - LB_j) \times \mu + LB_j) & rand > 0.5 \\ best(x_j) + MOP \times ((UB_j - LB_j) \times \mu + LB_j) & rand \leq 0.5 \end{cases} \tag{14}$$

3.6. Markov chain integration for VM load forecasting

In our proposed scheduling framework, Markov chains predict the future load state of virtual machines (VMs) in the fog-cloud environment. Each VM is categorized into two states: high load (H) and low load (L). The Markov chain employs a transition matrix to estimate the transition probabilities between these states. This predictive capability informs scheduling decisions, enabling the allocation of tasks to VMs that are anticipated to have lower loads, thereby minimizing delays and avoiding resource bottlenecks. The integration of Markov chains enhances the overall efficiency of the Arithmetic Optimization Algorithm (AOA). During each iteration of the scheduling process, AOA considers the predicted VM load states alongside task deadlines and resource availability. The scheduling objective function incorporates traditional metrics, such as makespan and energy consumption, while factoring in load balancing based on the predicted VM states. This approach allows the algorithm to adapt to changes in resource demand dynamically, ensuring that tasks are allocated in a manner that optimizes resource usage and minimizes task completion times. By predicting which VMs will experience high or low loads shortly, the system dynamically adjusts task assignments to avoid overloading VMs and to ensure efficient resource utilization. This integration ultimately enhances load balancing, reduces task completion time (makespan), and minimizes delays by preventing VM overloading and underutilization.

3.7. Task classification and prioritization in scheduling

In our proposed framework, tasks are systematically classified based on urgency, resource requirements, and dependencies to facilitate efficient allocation within heterogeneous fog computing environments. Tasks are categorized as real-time or non-real-time. Real-time tasks, characterized by stringent deadlines and the need for immediate processing, are exemplified by applications in autonomous vehicles, industrial automation, and real-time healthcare monitoring, where processing delays can have significant consequences. Non-real-time tasks, such as background data processing or periodic data aggregation, are less time-sensitive and can tolerate some delay. Further classification is based on resource requirements, distinguishing between lightweight and heavyweight tasks. Lightweight tasks, requiring minimal computational resources, are suitable for execution on edge devices, while heavyweight tasks, demanding substantial processing power and memory, often necessitate allocation to more capable fog nodes or cloud resources. Finally, tasks are identified as independent or dependent based on execution relationships. Independent tasks can be executed in isolation, whereas dependent tasks require a specific execution order due to their reliance on outputs from preceding tasks. This comprehensive classification enables effective prioritization, ensuring that high-priority tasks are allocated resources first, thereby optimizing overall system performance.

The task allocation decision-making process is dynamic and responsive to the heterogeneous nature of fog computing resources, which vary in processing power, memory capacity, and network bandwidth. Real-time tasks are strategically allocated to the nearest fog nodes to minimize latency and ensure swift processing. Conversely, non-real-time tasks are assigned to cloud resources or lower-priority fog nodes, balancing the system load and maximizing resource utilization. Integrating Markov chains enhances predictive load balancing by forecasting the load states of virtual machines (VMs) and fog nodes, enabling dynamic task migration and redistribution. For example, suppose a VM is predicted to experience a high load. In that case, tasks can be seamlessly transferred to underutilized VMs or fog nodes, maintaining even workload distribution and preventing resource congestion.

Furthermore, the Arithmetic Optimization Algorithm (AOA) refines the task-to-resource mapping, considering each task's specific resource requirements and the capabilities of available resources. This integration minimizes makespan and energy consumption while enhancing system responsiveness and efficiency. This comprehensive approach to task classification, prioritization, and allocation effectively addresses the challenges of heterogeneous fog computing environments.

3.8. Flowchart of the proposed method

Fig. 5 shows the flowchart of the proposed method for scheduling and resource allocation. It has the following stages:

- Creating an initial population of task-to-resource coding vectors
- Evaluating solutions with the objective function and selecting the optimal task-to-resource mapping vector
- Updating task mappings with four operators: addition, subtraction, multiplication, and division
- Selecting the optimal mapping in the final iteration and scheduling based on it.

4. Experimental

The simulations of the proposed method and other algorithms in this paper were conducted using MATLAB R2021b software. An Asus laptop with a Core i7 processor, 16 GB of RAM, a 2.50 GHz clock speed, and a 64-bit Windows operating system was used. The NASA Ames iPSC/860 dataset contains information on 14,794 jobs and tasks. For the NASA iPSC dataset, which involves 257 users, 240 processors or CPUs are required. The average number of tasks in this dataset is 202.871. Initially, the evaluation parameters are reviewed, followed by the presentation of the evaluation indicators and experiments. In the final stage, the results of the proposed method will be compared with similar techniques.

4.1. Parameters

The parameters of the proposed method need to be carefully configured to implement it. In the implementations, first, it is tried to set the AOA algorithm parameters accurately. In the experiments, the population size in the AOA algorithm is set to 10, and the number of iterations is set to 30. Each experiment is repeated 50 times, and the

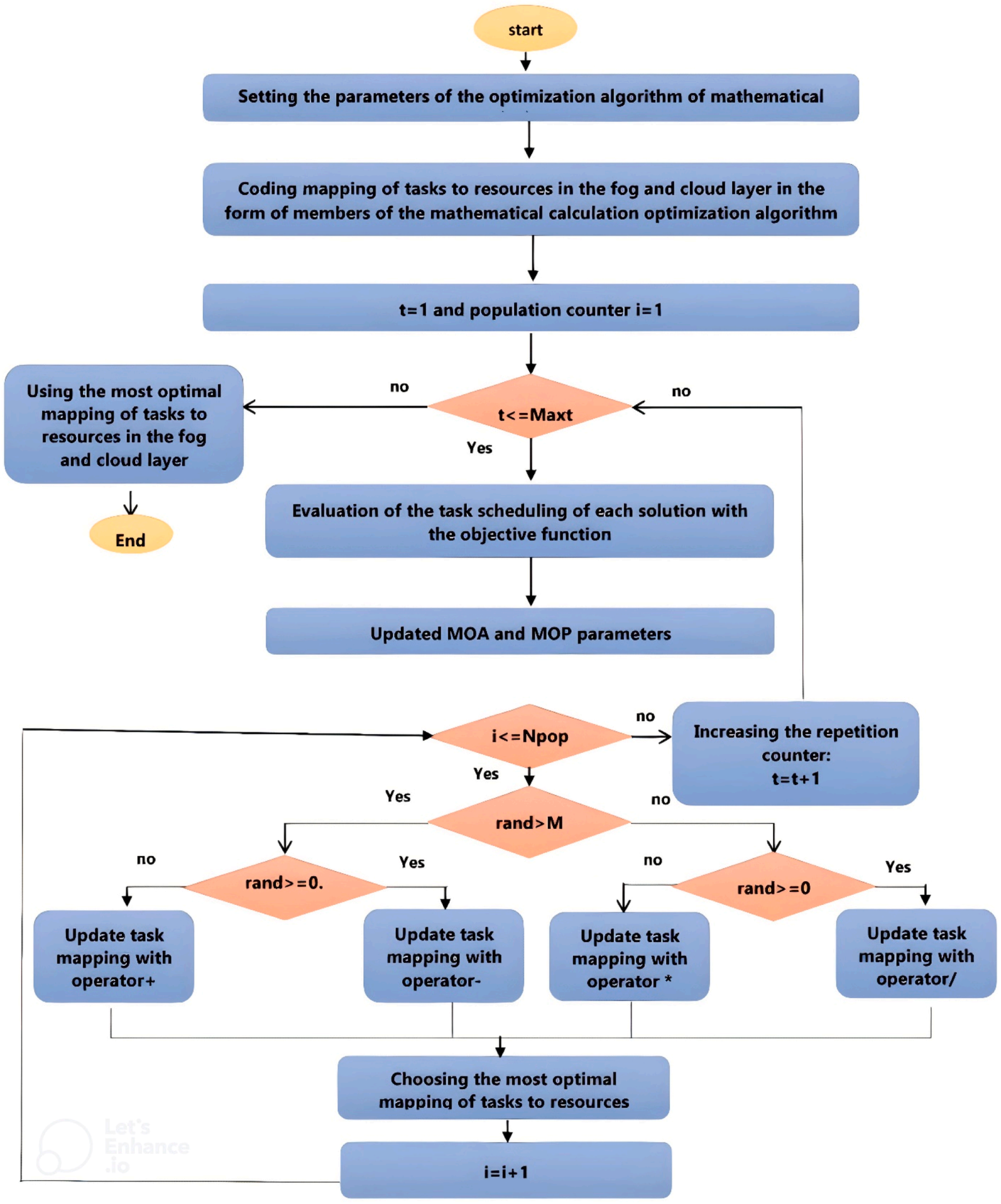
start
Setting the parameters of the optimization algorithm of mathematical
Coding mapping of tasks to resources in the fog and cloud layer in the form of members of the mathematical calculation optimization algorithm
t=1 and population counter i=1
t<=Maxt
no
Yes
no
Using the most optimal mapping of tasks to resources in the fog and cloud layer
End
Evaluation of the task scheduling of each solution with the objective function
Updated MOA and MOP parameters
i<=Npop
no
Increasing the repetition counter: t=t+1
Yes
rand>M
Yes
no
rand>=0.
no
Yes
rand>=0
no
Yes
Update task mapping with operator+
Update task mapping with operator-
Update task mapping with operator *
Update task mapping with operator/
Choosing the most optimal mapping of tasks to resources
i=i+1

Fig. 5. Flowchart of the proposed method in resource allocation and scheduling.

average evaluation indicators are considered for analysis. The α parameter in the mathematical optimization algorithm is set to 5, and in the AOA algorithm, ε is set to a minimal value, approximately 0.001. In implementing the proposed method in the cloud layer, the number of data centres used is equal to 2, and the number of hosts is 4. The number of virtual machines in each host is set to 10, and the processing power of each machine is considered between 3000 and 5000 MIPS. The main memory of each VM is set to 1 GB, and the secondary memory of the virtual machine is set to 20 GB. The computational bandwidth capacity of virtual machines is 0.5 GB per second. The number of VMs in the fog nodes is 15, and the processor power in the fog layer is between 100 and 2800 MIPS. The main memory of fog nodes is 0.5 GB, the secondary memory is 10 GB, and the data transfer rate in the fog layer is 1 GB per second, with each fog node having one CPU.

4.2. Implementation metrics

In evaluating the proposed method, makespan is the most critical evaluation metric for task scheduling and resource allocation. This metric indicates the time at which the last task finishes. A reduction in this parameter indicates better quality of the proposed method. The second important metric is throughput, which refers to the total number of tasks successfully executed within a specified time frame. To calculate the throughput metric, Eq. (15) can be used:

$$Throughput = \sum_{t_i \in Tasks} ExeTime(t_i) \tag{15}$$

Y' represents the fitness obtained by the proposed method in scheduling, and Y represents the fitness value received by each comparison method in scheduling.

4.3. Time complexity analysis

Resource allocation and scheduling algorithms are very time-sensitive, and a proper scheduling algorithm should have low complexity. In this section, the proposed method is compared with well-known meta-heuristic algorithms such as Particle Swarm Optimization (PSO), Firefly Algorithm (FA), Whale Optimization Algorithm (WOA), Hawk Optimization Algorithm (HOA), and Jellyfish Optimization Algorithm (JOA). It should be noted that these methods are coded in the MATLAB environment and compared with the proposed method. The execution times of scheduling algorithms with the proposed method for 500, 1000, and 1500 tasks are compared in Figs. (6), (7), and (8).

To facilitate direct comparison with the results presented in reference [4], initial simulations were conducted with task counts of 500, 1000, and 1500. To further evaluate the robustness and scalability of our proposed method, the simulation scope was expanded to include task counts of 2000 and 2500. Table 3 presents the execution times for these simulations. The table shows that the proposed method consistently outperforms ChOA, WOA, HGSWC, and CSA across all task counts, demonstrating its superior performance in handling varying workloads.

Experimental analysis shows that if the number of tasks is 500, the execution time of task scheduling in the proposed method is 6.86 seconds.

In contrast, the execution times for PSO, FA, WOA, HOA, and JOA are 8.96, 7.46, 10.98, 15.43, and 11.56 seconds, respectively. In this case, the proposed method has a lower execution time than similar algorithms. With the increase in the number of tasks from 500 to 1000, the execution time of tasks will increase, and the execution time of functions in the proposed method will be 13.41 seconds. With the increase in the number of tasks to 1000, the execution times for PSO, FA, WOA, HOA, and JOA are 15.63, 14.72, 21.34, 30.84, and 23.14 seconds, respectively. If the number of tasks increases to 1500, the scheduling time for tasks in the proposed method will be 19.21 seconds, less than the execution time required by PSO, FA, WOA, HOA, and JOA algorithms. The analysis of average scheduling time behaviour is also precious, and in Fig. (9), the average execution time of the proposed algorithm is compared with other algorithms.

Experiments showed that the average execution time index of the proposed algorithm is 13.16 seconds, while the average execution time of PSO, FA, WOA, HOA, and JOA is 16.92, 14.35, 20.72, 29.82, and 22.96 seconds, respectively. Evaluations show that the proposed algorithm requires less execution time than other meta-heuristic algorithms, with the FA in second place. The worst performance among the scheduling algorithms is related to the HHO due to its higher computational complexity.

4.4. PIR index analysis

In this section, the effectiveness of the proposed algorithm based on the PIR index compared to meta-heuristic algorithms is compared in Fig. (10).

The PIR index indicates how successful the proposed algorithm has been in reducing the execution time and scheduling tasks compared to each meta-heuristic method. According to the experiments, the PIR index of the proposed method shows improvements of 22.22 %, 8.29 %, 36.48 %, 55.86 %, and 9.8 % compared to PSO, FA, WOA, HOA, and JOA algorithms, respectively.

4.5. Delay index analysis

This section compares the proposed algorithm with various optimization and scheduling methods based on the delay index. In Figs. (11), (12), (13), (14), and (15), the scheduling delay for 30, 50, 90, 150, and

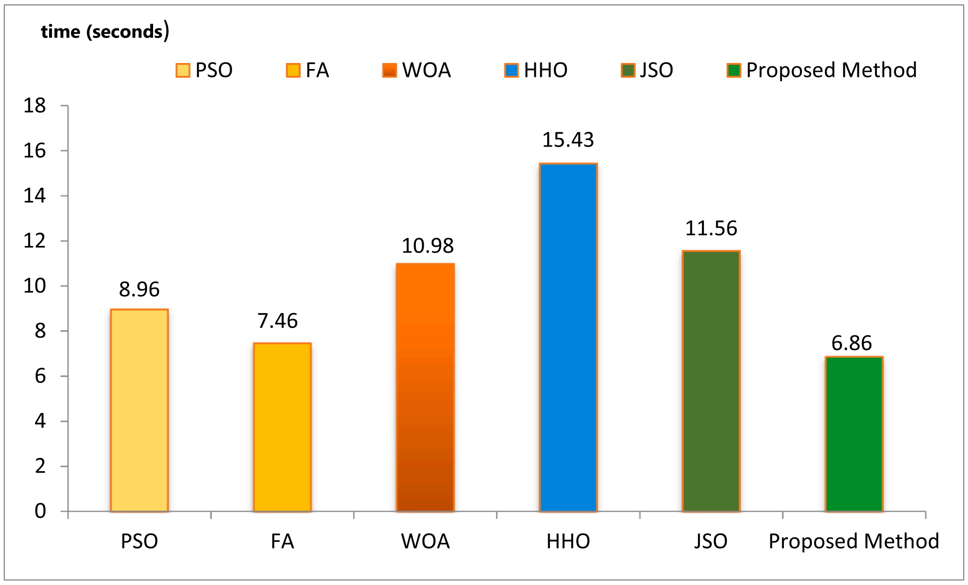

Fig. 6. Comparing the execution time of the proposed algorithm with meta-heuristic methods with 500 tasks.

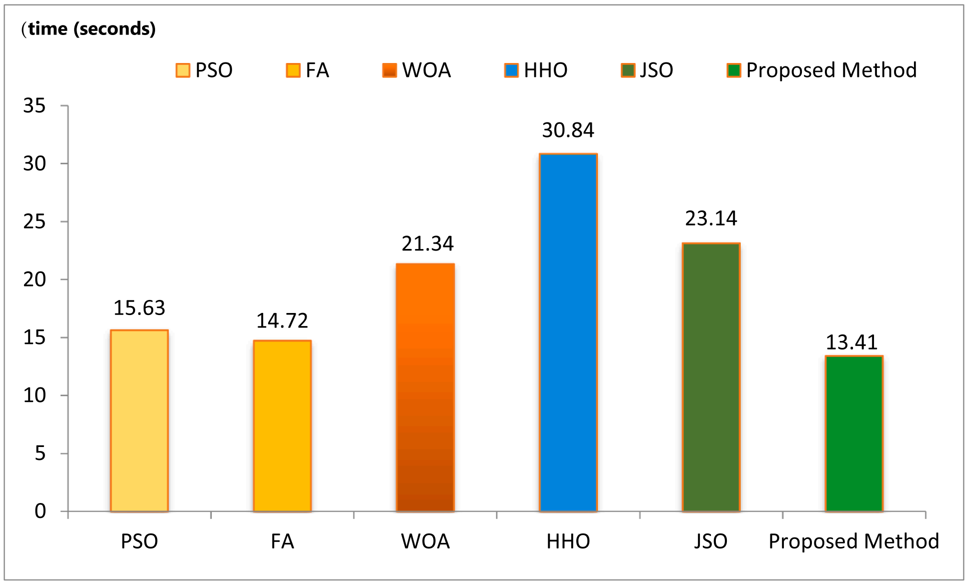

Fig. 7. Comparing the execution time of the proposed algorithm with meta-heuristic methods with 1000 tasks.

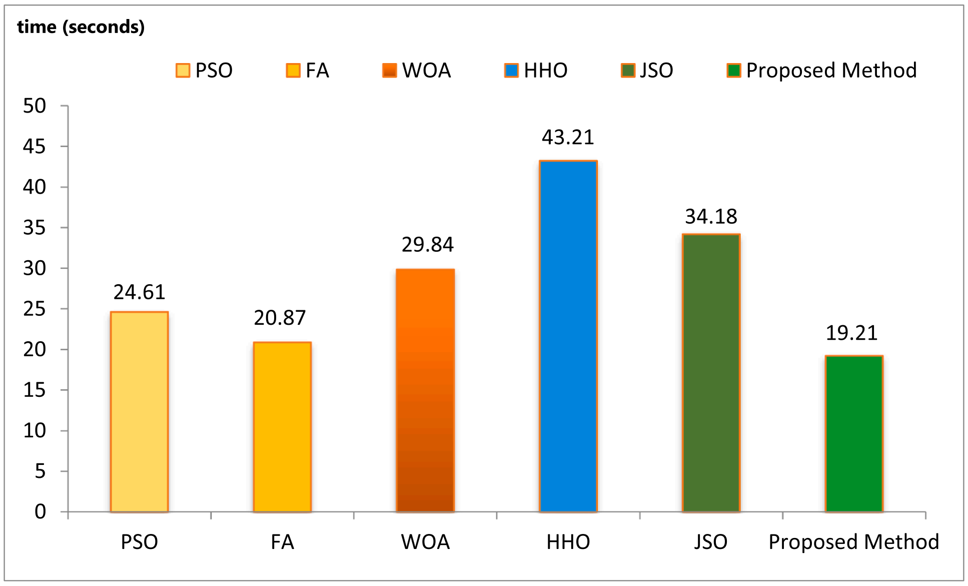

Fig. 8. Comparing the execution time of the proposed algorithm with meta-heuristic methods with 1500 tasks.

Table 3
Comparison of the execution time on different tasks.

Tasks	CSA	WOA	HGSWC	ChOA	CHMPAD	Proposed Method
500	55.09	71.53	58.45	65.70	51.78	47.21
1000	106.67	142.66	116.52	128.18	100.06	96.83
1500	157.99	205.26	178.16	182.64	147.90	130.38
2000	207.24	276.32	235.84	244.14	195.49	138.21
2500	257.85	350.09	299.53	296.97	244.09	227.53

200 tasks is compared with the proposed method and the results of research [4] are considered for comparison. In these experiments, the proposed method is compared with scheduling methods such as Cloud-Fog, MGWO, NSGA-II, and Multiple Objective Particle Swarm Optimization (MOPSO).

In the first experiment, the number of tasks is 30, and the delay for these 30 tasks is 20.82 seconds. The delay per task is calculated by dividing 20.82 by 30, which is 0.694 seconds.

If the number of tasks is 50, the delay of the proposed method is 39.71 seconds, which is less than scheduling methods such as Cloud-Fog, MGWO, NSGA-II, and MOPSO.

With the increase in the number of tasks to 150 and 200, the execution delay of tasks will rise in all algorithms, including the proposed method, and the following results are provided:

- For different numbers of tasks, the proposed method has less scheduling delay compared to Cloud-Fog, MGWO, NSGA-II, and MOPSO methods.
- With the growth in the number of tasks entering the fog and cloud layer, the delay caused by allocation and scheduling will increase.
- Due to simultaneous exploration and local search performance, the proposed method has more resource allocation and task scheduling capability than a genetic algorithm, PSO, and grey wolf optimization.

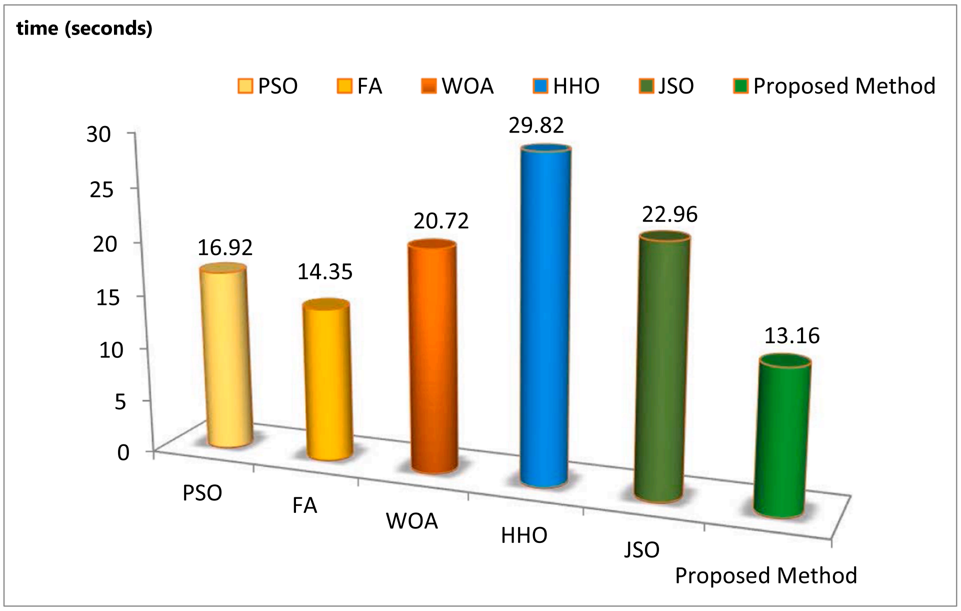

Fig. 9. Comparison of the average execution time of the proposed algorithm with meta-heuristic methods with different tasks.

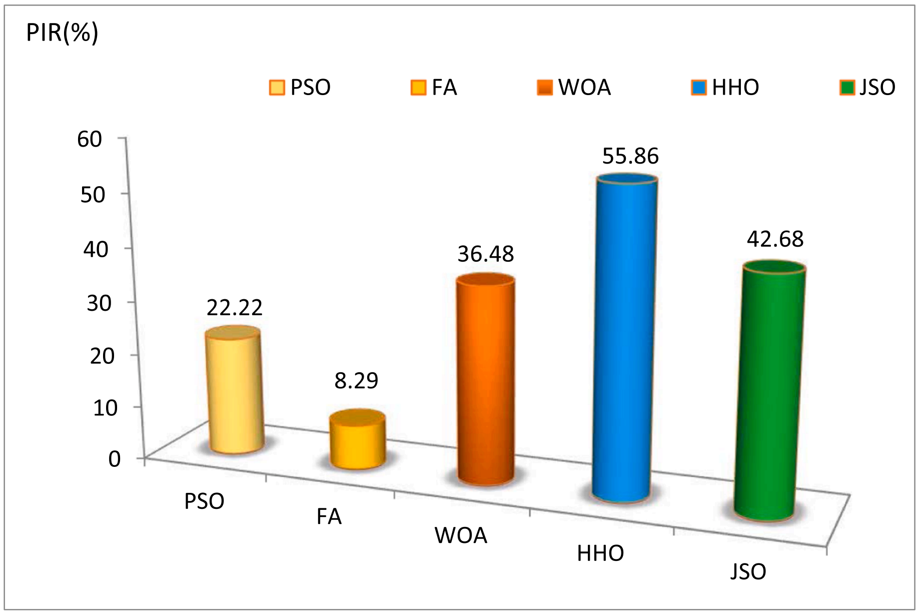

Fig. 10. Comparing the improvement of the execution time of the proposed algorithm with meta-heuristic methods in allocation and scheduling.

- The proposed algorithm has more balance and stability than similar methods because the delay caused by task allocation and task scheduling in different states is approximately 0.6–0.8 seconds.
- The grey wolf optimization algorithm has more delay than PSO and genetic algorithm.

4.6. *Makespan index analysis*

This section compares the proposed algorithm with meta-heuristic methods for scheduling using the NASA iPSC dataset [33]. In Figs. (16), (17), and (18), the suggested method is compared with base meta-heuristic methods such as ChOA (Chimp Optimization Algorithm), WOA, HGSWC (Henry Gas Solubility Optimization & WOA), and CSA (Crow Search Algorithm) in terms of the makespan. The number of tasks is set to 500, 1000, and 1500. In the graph shown in Figs. (4–15), the makespan for the proposed method, ChOA, WOA, HGSWC, and CSA, are 47.21, 65.70, 58.45, 71.53, and 55.09, respectively. The proposed method completes all tasks earlier than the similar methods for 500 tasks. In Figs. (4–16) and (4–17), it is observed that the task execution time in the proposed method is completed faster. This is due to the more intelligent search mechanism of the proposed method. The Crow Search Algorithm (CSA) is more successful among the compared methods after the proposed algorithm.

Based on the charts and simulation results, the makespan, delay, and PIR rates have improved compared to some existing methods. The reason for this improvement is that due to the repetitive nature of the mathematical and meta-heuristic optimization algorithm (HOA), the delay continuously decreases, which is related to the improvement in task-to-resource mapping. Additionally, using the Markov chain has led to tasks being allocated to resources with lower computational loads, enhancing the quality of service. Finally, the simulation's simultaneous usage of fog and cloud resources ensures real-time tasks are executed in

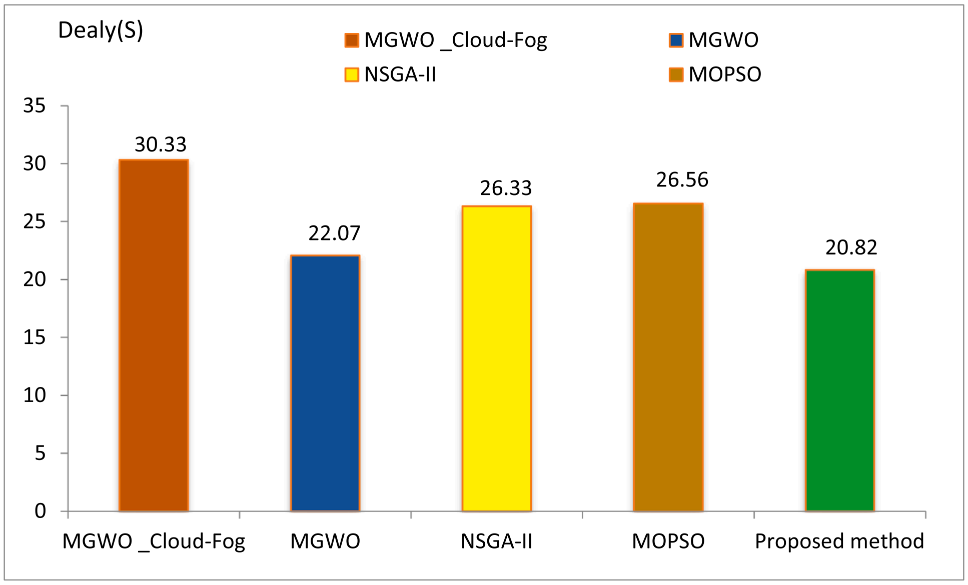

Fig. 11. Comparison of allocation and scheduling delays for 30 tasks.

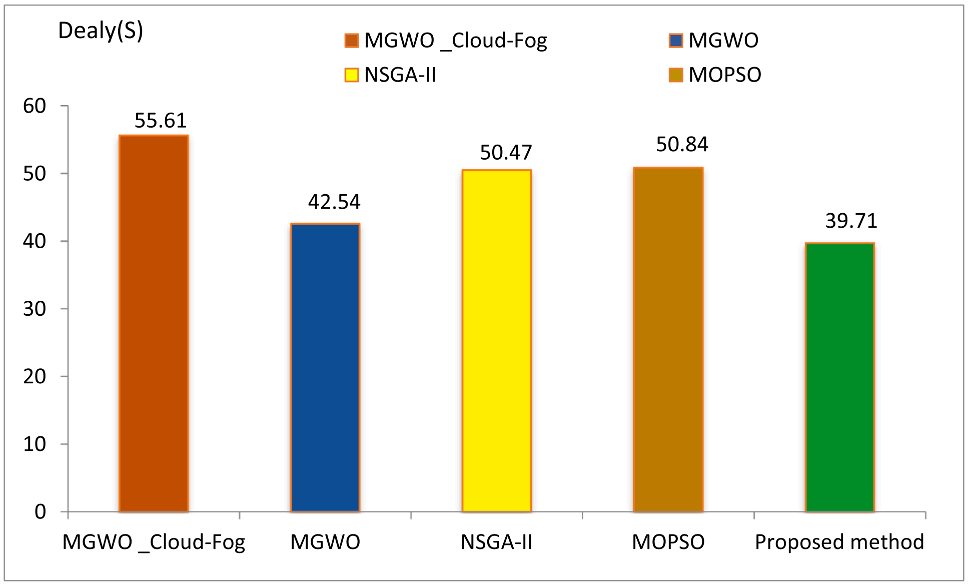

Fig. 12. Comparison of allocation and scheduling delays for 50 tasks.

the fog layer. In contrast, non-real-time tasks are allocated resources from the cloud layer. This reduces delay and consequently decreases energy consumption.

5. Results and discussion

In fog computing environments, the core challenges of load instability, excessive makespan, and underutilization of virtual machines significantly impact the performance of IoT systems. Load instability leads to unpredictable resource bottlenecks, causing delays in applications such as intelligent city traffic management, healthcare monitoring, and industrial IoT systems. Makespan inefficiencies result in delayed task completion, with critical implications for applications requiring real-time processing, such as autonomous vehicles, real-time video analytics, and e-commerce platforms. Furthermore, the underutilization of virtual machines wastes computational resources and increases operational costs, particularly in domains like telecommunications, cloud gaming, and renewable energy management.

The performance of the proposed HMCDS framework was evaluated using key metrics, including execution time, Performance Improvement Ratio (PIR), makespan, delay, and energy consumption. These metrics were compared against several established algorithms: Particle Swarm Optimization (PSO), Firefly Algorithm (FA), Whale Optimization Algorithm (WOA), and Grey Wolf Optimization (GWO). The results demonstrated that HMCDS consistently outperformed the comparative algorithms across all metrics. Notably, HMCDS achieved the shortest execution times across all tested task sets (500, 1000, and 1500 tasks), with this performance advantage becoming more pronounced as the number of tasks increased, highlighting its scalability and efficiency in handling larger workloads. Furthermore, HMCDS exhibited a significant Performance Improvement Ratio (PIR) compared to other algorithms, achieving a 22.22 % improvement in execution time and an 8.29 % improvement in makespan compared to PSO.

HMCDS consistently achieved the lowest values regarding makespan and delay, indicating its superior scheduling efficiency. These results underscore HMCDS's suitability for real-time applications, where timely

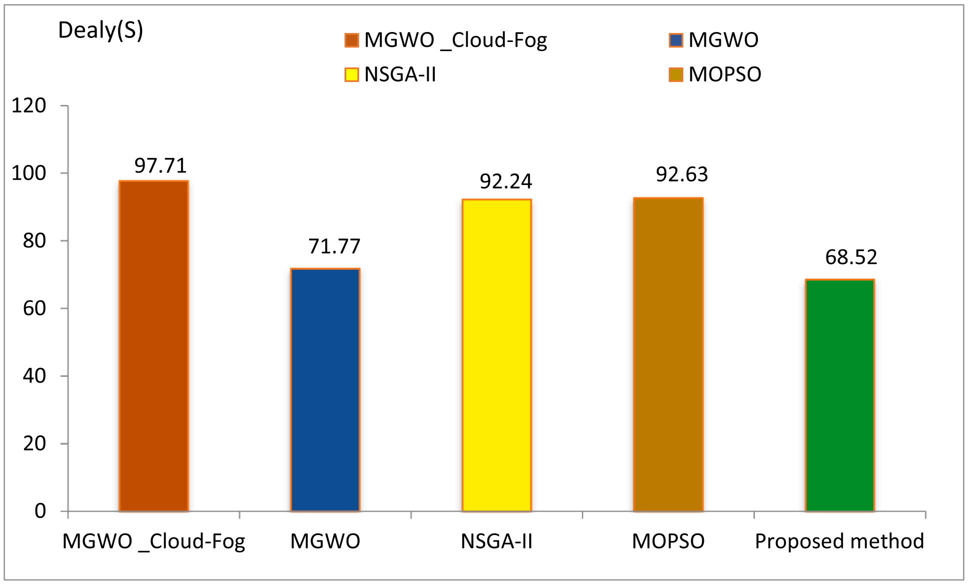

Fig. 13. Comparison of allocation and scheduling delays for 90 tasks.

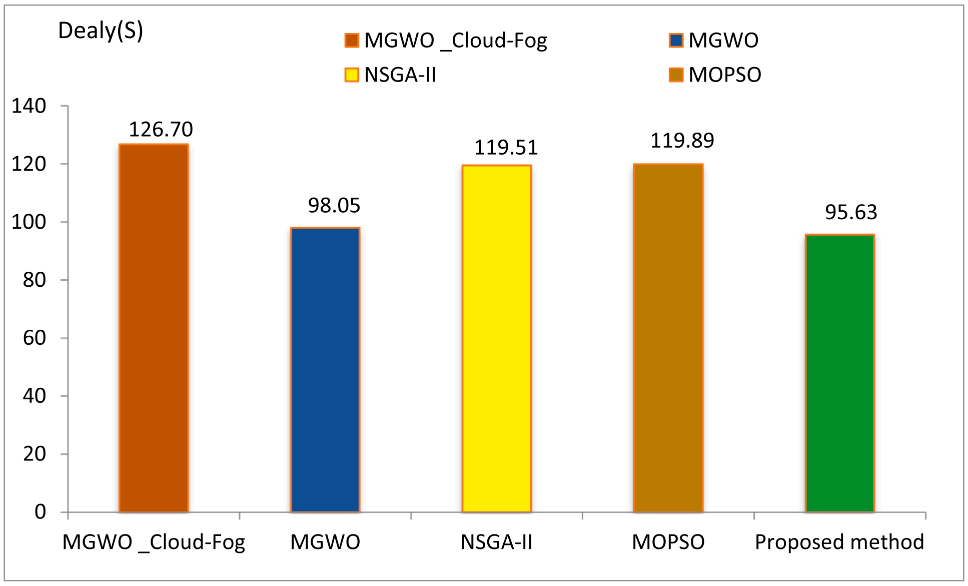

Fig. 14. Comparison of allocation and scheduling delays for 150 tasks.

task completion is crucial. Moreover, HMCDS exhibited the lowest energy consumption among all algorithms. This can be attributed to its efficient task-to-resource mapping and dynamic load balancing, which minimize idle time and avoid overburdened resources, thereby reducing energy consumption. These findings demonstrate that integrating Markov chain-based load prediction with the Arithmetic Optimization Algorithm (AOA) within the HMCDS framework provides several key advantages. Firstly, HMCDS significantly reduces makespan and delay, making it well-suited for real-time applications in IoT environments, such as autonomous vehicles and innovative healthcare. Secondly, by optimizing resource utilization and minimizing idle time, HMCDS reduces energy consumption, which is crucial for resource-constrained fog nodes and battery-powered devices. Finally, HMCDS demonstrates scalability in handling increasing task volumes, making it suitable for growing IoT ecosystems with many devices and tasks.

The observed performance improvements, including the 8.29 % improvement in makespan, can be attributed to the superior search capabilities of the AOA compared to other metaheuristic algorithms. For instance, the Firefly Algorithm (FA) employs a solution attraction mechanism but lacks a balanced approach between local and global search. This imbalance makes FA prone to becoming trapped in local optima, limiting its ability to find globally optimal solutions. Furthermore, FA relies on a single central equation for modelling, whereas AOA utilizes four equations, demonstrating significantly higher modelling power and greater accuracy. The Crow Algorithm, while capable of storing the positions of optimal solutions, needs a better modelled local search behaviour. This deficiency prevents the Crow Algorithm from improving its accuracy in later iterations.

Additionally, memory consumption in the Crow Algorithm is higher than that of AOA, as AOA does not require the history of all solutions to be stored. Finally, the Grey Wolf Optimization (GWO) algorithm heavily depends on its three best solutions. The algorithm can be misled if these solutions converge near a local optimum, leading to suboptimal results. In contrast, AOA employs operators such as division and multiplication to effectively escape local optima, allowing it to explore a more expansive search space and converge towards better solutions. This

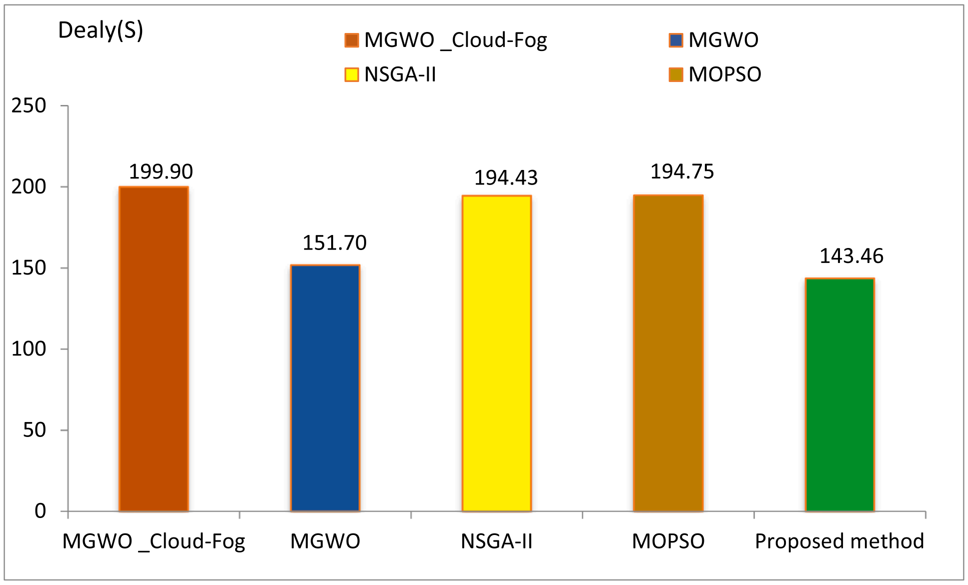

Fig. 15. Comparison of allocation and scheduling delays for 200 tasks.

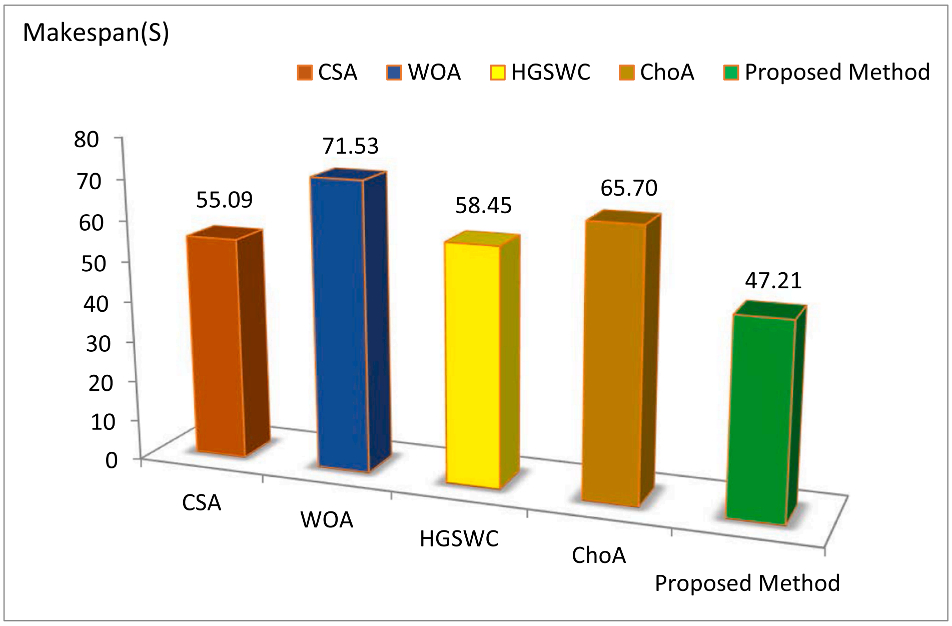

Fig. 16. Comparison of Makespan in allocation and scheduling for 500 tasks.

ability to escape local optima is a key factor in AOA's improved performance and contributes to the overall effectiveness of the HMCDS framework.

Beyond energy consumption and delay, other crucial performance aspects in fog computing environments include resource utilization, scalability, and fault tolerance, which we now discuss. Our proposed optimization approach inherently aims to optimize resource usage. The AOA algorithm's effective resource allocation across processors, memory, and virtual machines directly contributes to the observed reduction in energy consumption. Scalability is another key consideration, ensuring the algorithm's effective operation regardless of the number of resources. The distributed architecture of cloud and fog environments necessitates handling resources across various nodes, and the AOA algorithm is designed to select optimal resource arrangements from these distributed pools. Within metaheuristic algorithms, each resource mapping and arrangement represents a problem solution executable in various dimensions, thus demonstrating inherent scalability. However, it is essential to acknowledge that the computational time required by the metaheuristic algorithm increases with the number of fog nodes and candidate resources.

Regarding fault tolerance, the proposed method performs adequately if the algorithm avoids local optima and does not produce suboptimal solutions. If scheduled resources become unavailable during the AOA scheduling process, it will result in an error. Therefore, the method's fault tolerance is acceptable when the scheduled resources remain available.

While the proposed HMCDS framework effectively reduces energy usage and delay, it is essential to consider potential trade-offs. Integrating the Arithmetic Optimization Algorithm (AOA) and the Markov chain-based load prediction introduces additional computational overhead compared to more straightforward scheduling approaches. However, as shown in Fig. 7, the execution time of HMCDS for 1000 tasks is

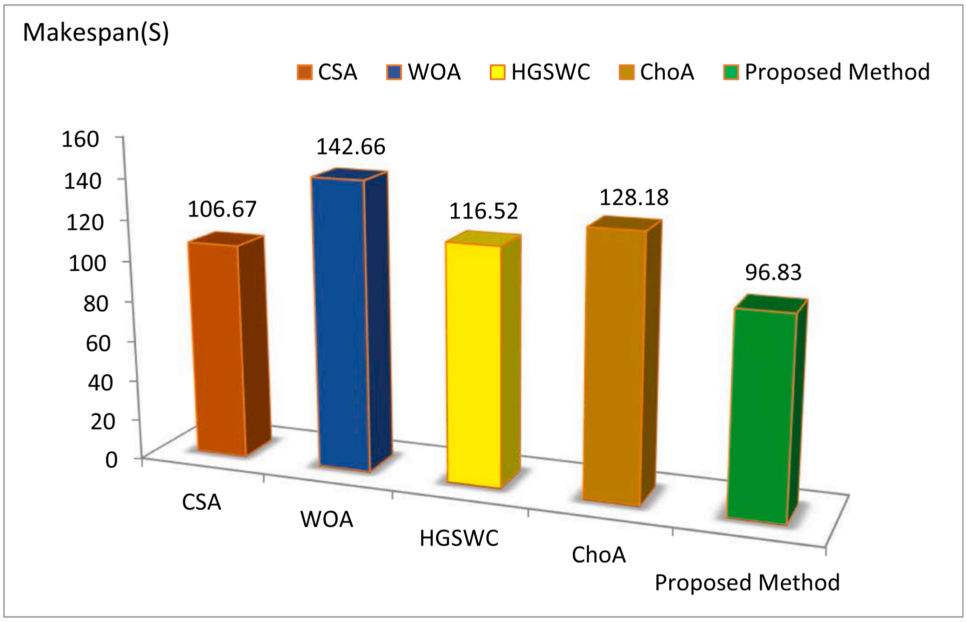

Fig. 17. Comparison of Makespan in allocation and scheduling for 1000 tasks.

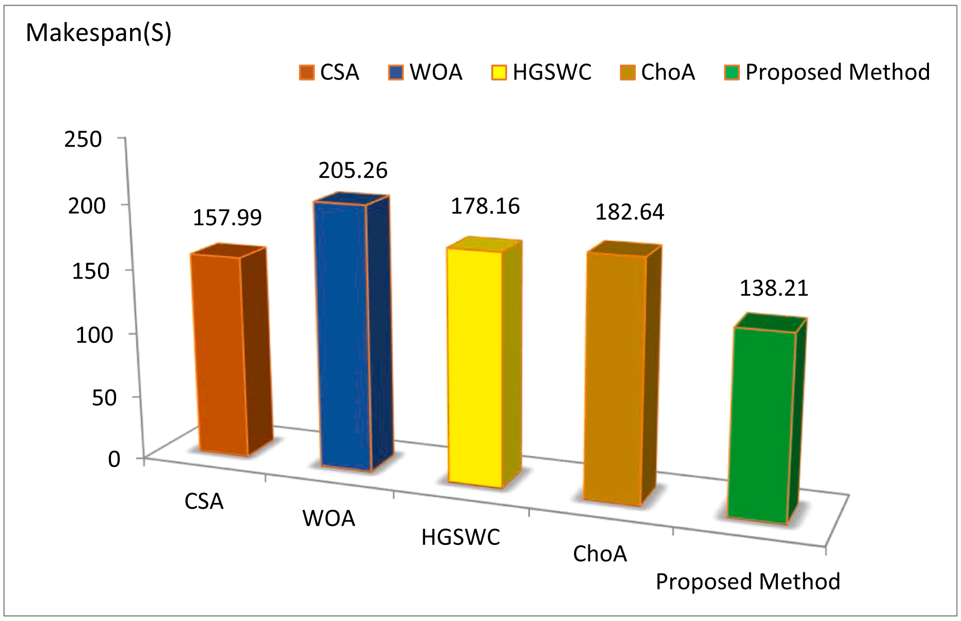

Fig. 18. Comparison of Makespan in allocation and scheduling for 2000 tasks.

13.41 seconds, which is comparable to or lower than that of other metaheuristic algorithms like PSO (15.63 seconds), FA (14.72 seconds), WOA (21.34 seconds), HHO (30.84 seconds), and JSO (23.14 seconds). Moreover, the computational overhead of the Markov chain is relatively low due to its memoryless property, making it less complex than machine learning and deep learning-based approaches. Further analysis, as shown in Fig. 19, reveals that when executing 500 tasks, the Markov algorithm alone takes 2.45 seconds, the AOA algorithm takes 4.34 seconds, and the combined HMCDS approach takes 6.86 seconds. As the number of tasks increases to 1000 or 1500, the execution time for all three cases (Markov, AOA, and HMCDS) naturally increases. However, while the combined approach incurs a higher computational cost than using the Markov chain or AOA alone, this increase is justified by the significant reduction in energy consumption achieved across both fog and cloud layers, which is a primary objective of this study.

6. Conclusion

In this paper, we proposed the Hybrid Markov Chain-Based Dynamic Scheduling (HMCDS) framework for efficient task scheduling in fog-cloud environments. HMCDS leverages the combined resources of both fog and cloud layers, employing Markov chain-based load prediction to anticipate resource demands and the Arithmetic Optimization Algorithm (AOA) for optimal task-to-resource mapping. This combined approach aims to improve resource allocation efficiency, minimize task completion delays, and enhance overall makespan. Through comprehensive simulations, we demonstrated that HMCDS outperforms existing metaheuristic methods such as Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Grey Wolf Optimization (GWO), particularly in minimizing delay and achieving more effective resource allocation. However, the proposed approach has limitations. The inherent

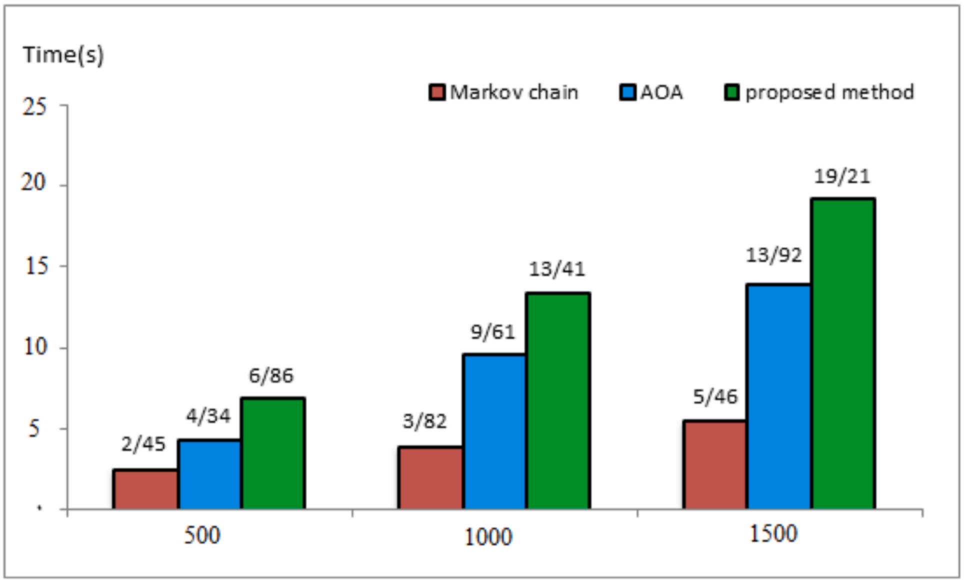

Fig. 19. Comparison of the execution time of the proposed algorithm in three scenarios with 500, 1000, and 1500 tasks.

computational complexity of both the AOA and the Markov chain-based prediction mechanism can pose challenges to scalability in large-scale IoT deployments with a high volume of tasks and fog nodes. This complexity could increase processing overhead and lead to longer decision-making times as the system scales.

Furthermore, the accuracy of the Markov chain's load predictions can be susceptible to sudden and unpredictable fluctuations in workload, potentially leading to suboptimal task allocations and reduced overall system performance. Future work will focus on enhancing HMCDS by improving load prediction accuracy through the exploration of hybrid prediction models incorporating machine learning techniques and real-time adaptation mechanisms. We will also investigate approximation methods and parallel processing techniques to address the scalability challenges and extend the framework's application to specific IoT use cases, such as competent healthcare, autonomous vehicles, and industrial IoT, as well as integrating it with hybrid cloud-edge environments.

Enhanced literature review

o The literature review section has been reorganized into a tabular format, synthesizing key aspects of existing studies, including methodologies, advantages, and limitations. This format facilitates better comprehension and provides a clearer context for our contributions. (Section 2, Page 4)

1. **Clarity and Consistency**:
   o Abbreviations have been consistently defined and used throughout the manuscript to enhance readability.
   o Parameters, data, and results have been reorganized into well-structured tables to improve clarity. For instance, key parameters are presented in Table 2. (Section 3.1, Page 7)
2. **Expanded Methodology**:
   o Further details on the integration of the Markov chain for VM load forecasting and its role in the scheduling framework have been added. This includes an explanation of how task classification, prioritization, and allocation are managed in heterogeneous fog environments. (Section 3.6, Page 17; Section 3.7, Page 16)
3. **Motivation and Novelty**:
   o The introduction has been revised to clearly articulate the significance of using the Arithmetic Optimization Algorithm (AOA) and the research gaps it addresses in task scheduling. (Section 1, Page 2)
4. **Comparison with Existing Approaches**:
   o A dedicated comparison table (Table 1) has been included to juxtapose the proposed method with existing algorithms, highlighting its novelty and efficacy. (Section 2, Page 4)
5. **Results and Discussion**:
   o This section has been expanded to provide a deeper analysis of the findings, including an explanation of the observed performance improvements and their implications. Metrics such as resource utilization, scalability, and fault tolerance have also been discussed. (Section 5, Page 27)
6. **Experimental Details**:
   o Information on the simulation environment, dataset, and system configuration has been added to enhance the reproducibility of the results. (Section 4, Page 17)
7. **Trade-offs and Limitations**:
   o A discussion of the potential trade-offs, such as computational overhead introduced by the AOA and Markov chain integration, has been included. This ensures a balanced view of the framework's strengths and limitations. (Results and Discussion, Page 29)
8. **Improved Visuals**:
   o The flowchart representing the proposed method (Fig. 5) has been revised for enhanced clarity and visual appeal. (Section 4.1, Page 17)
9. **Conclusion and Future Work**:
   o The conclusion has been revised to include a reflection on the limitations of the current approach and potential future research directions, such as extending the method to diverse IoT use cases and integrating with other hybrid cloud-edge frameworks. (Section 6, Page 29)

We believe that the above revisions comprehensively address the reviewers' comments and significantly improve the manuscript. We hope the revised version meets the expectations of the reviewers and the editorial team. Thank you for the opportunity to improve our work.

Author Statement

We would like to express our sincere gratitude to the reviewers and editors for their insightful feedback on our manuscript titled *"Hybrid Markov Chain-Based Dynamic Scheduling to Improve Load Balancing*

Performance in Fog-Cloud Environment." Their constructive comments have significantly contributed to improving the clarity, structure, and overall quality of our work. Below, we summarize the major changes made to the manuscript in response to their suggestions:

CRediT authorship contribution statement

Navid Khaledian: Writing – review & editing, Data curation, Conceptualization. **Zeynab Haghbayan:** Visualization, Software, Methodology. **Shiva Razaghzadeh:** Conceptualization, Supervision, data analysis, and editing. **Marcus Voelp:** Writing – review & editing, Validation, Formal analysis.

Declaration of Competing Interest

The author declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We gratefully acknowledge Fatemeh Poormohammad for their assistance in data analysis, programming, and manuscript editing.

Data availability

Data will be made available on request.

References

[1] Andrés Felipe Solis Pino, et al., Systematic literature review on mechanisms to measure the technological maturity of the Internet of Things in enterprises, Internet Things (2024) 101082.
[2] Navid Khaledian, et al., AI-based & heuristic workflow scheduling in cloud and fog computing: a systematic review, Clust. Comput. (2024) 1–34.
[3] Zhiyu Wang, et al., Deep Reinforcement Learning-based scheduling for optimizing system load and response time in edge and fog computing environments, Future Gener. Comput. Syst. 152 (2024) 55–69.
[4] Pakmehr, Amir, Majid Gholipour, and Esmaeil Zeinali ETFC: Energy-efficient and deadline-aware task scheduling in fog computing." *Sustainable Computing: Informatics and Systems* 43 (2024): 100988. Ali, Asad, et al. "Multi-Objective Harris Hawks Optimization Based Task Scheduling in Cloud-Fog Computing." *IEEE Internet of Things Journal* (2024).
[5] Asad Ali, et al., Multi-objective harris hawks optimization based task scheduling in cloud-fog computing, IEEE Internet Things J. (2024).
[6] Navid Khaledian, et al., An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment, computing 106 (1) (2024) 109–137.
[7] Reza Akraminejad, et al., A multi-objective crow search algorithm for optimizing makespan and costs in scientific cloud workflows (CSAMOMC), Computing (2024) 1–17.
[8] Nathaniel Hudson, et al., QoS-aware edge AI placement and scheduling with multiple implementations in FaaS-based edge computing, Future Gener. Comput. Syst. 157 (2024) 250–263.
[9] Sumathi Gurusamy, Rajesh Selvaraj, Resource allocation with efficient task scheduling in cloud computing using hierarchical auto-associative polynomial convolutional neural network, Expert Syst. Appl. (2024) 123554.
[10] Imane Aly Saroit, Tarek Dina, LBCC-Hung: a load balancing protocol for cloud computing based on Hungarian method, Egypt. Inform. J. 24 (3) (2023) 100387.
[11] Liu, Juan, et al. "Delay-optimal computation task scheduling for mobile-edge computing systems." 2016 IEEE international symposium on information theory (ISIT). IEEE, 2016.
[12] Ismail M. Ali, et al., An automated task scheduling model using non-dominated sorting genetic algorithm II for fog-cloud systems, IEEE Trans. Cloud Comput. 10 (4) (2020) 2294–2308.
[13] Ali Mohammadzadeh, et al., Energy-aware workflow scheduling in fog computing using a hybrid chaotic algorithm, J. Supercomput. 79 (16) (2023) 18569–18604.
[14] Gyan Singh, Amit K. Chaturvedi, Hybrid modified particle swarm optimization with genetic algorithm (GA) based workflow scheduling in cloud-fog environment for multi-objective optimization, Clust. Comput. 27 (2) (2024) 1947–1964.
[15] Navid Khaledian, et al., IKH-EFT: an improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment, Sustain. Comput.: Inform. Syst. 37 (2023) 100834.
[16] Fatemeh Ramezani Shahidani, et al., Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm, Computing 105 (6) (2023) 1337–1359.
[17] Seyyedamin Seifhosseini, Mirsaeid Hosseini Shirvani, Yaser Ramzanpoor, Multi-objective cost-aware bag-of-tasks scheduling optimization model for IoT applications running on heterogeneous fog environment, Comput. Netw. 240 (2024) 110161.
[18] Mirsaeid Hosseini Shirvani, Yaser Ramzanpoor, Multi-objective QoS-aware optimization for deployment of IoT applications on cloud and fog computing infrastructure, Neural Comput. Appl. 35 (26) (2023) 19581–19626.
[19] Shabina Ghafir, et al., Load balancing in cloud computing via intelligent PSO-based feedback controller, Sustain. Comput.: Inform. Syst. 41 (2024) 100948.
[20] Raj Mohan Singh, Geeta Sikka, Lalit Kumar Awasthi, LBATSM: load balancing aware task selection and migration approach in fog computing environment, IEEE Syst. J. (2024).
[21] Muhammad Ibrahim, YunJung Lee, Do-Hyuen Kim, DALBFog: deadline-aware and load-balanced task scheduling for the Internet of Things in fog computing, IEEE Syst., Man, Cybern. Mag. 10 (1) (2024) 62–71.
[22] Vijaita Kashyap, Rakesh Ahuja, Ashok Kumar, A hybrid approach for fault-tolerance aware load balancing in fog computing.", Clust. Comput. (2024) 1–17.
[23] Kiran, Koppolu Ravi, et al. An advanced ensemble load balancing approach for fog computing applications." *International Journal of Electrical & Computer Engineering* (2088-8708) 14.2 (2024).
[24] A. Ghorbannia Delavar, R. Akraminejad, WSTMOS: a method for optimizing throughput, energy, and latency in cloud workflow scheduling, J. Inf. Commun. Technol. 57 (57) (2023) 62.
[25] A.G. Delavar, R. Akraminejad, S. Mozafari, HDECO: a method for Decreasing energy and cost by using virtual machine migration by considering hybrid parameters, Comput. Commun. 195 (2022) 49–60.
[26] S. Kushwaha, R.S. Singh, Deadline and budget-constrained Archimedes optimization algorithm for workflow scheduling in cloud, Clust. Comput. 28 (2024) 117.
[27] Shahriar Karami, Sadoon Azizi, Fardin Ahmadizar, A bi-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization, Appl. Soft Comput. 151 (2024) 111142.
[28] R. Sing, S.K. Bhoi, N. Panigrahi, K.S. Sahoo, M. Bilal, S.C. Shah, EMCS: an energy-efficient makespan cost-aware scheduling algorithm using evolutionary learning approach for cloud-fog-based IoT applications, Sustainability 14 (22) (2022) 15096.
[29] I. Attiya, L. Abualigah, D. Elsadek, S.A. Chelloug, M. AbdElaziz, An intelligent chimp optimizer for scheduling of IoT application tasks in fog computing, Mathematics 10 (7) (2022) 1100.
[30] F.A. Saif, R. Latip, Z.M. Hanapi, K. Shafinah, Multi-objective grey wolf optimizer algorithm for task scheduling in cloud-fog computing, IEEE Access 11 (2023) 20635–20646.
[31] L. Liu, H. Xu, B. Wang, C. Ke, Multi-strategy fusion of sine cosine and arithmetic hybrid optimization algorithm, Electronics 12 (9) (2023) 1961.
[32] L. Abualigah, A. Diabat, S. Mirjalili, M. AbdElaziz, A.H. Gandomi, The arithmetic optimization algorithm, Comput. Methods Appl. Mech. Eng. 376 (2021) 113609.
[33] I. Attiya, L. Abualigah, D. Elsadek, S.A. Chelloug, M. AbdElaziz, An intelligent chimp optimizer for scheduling of IoT application tasks in fog computing, Mathematics 10 (7) (2022) 1100.